Figure 1
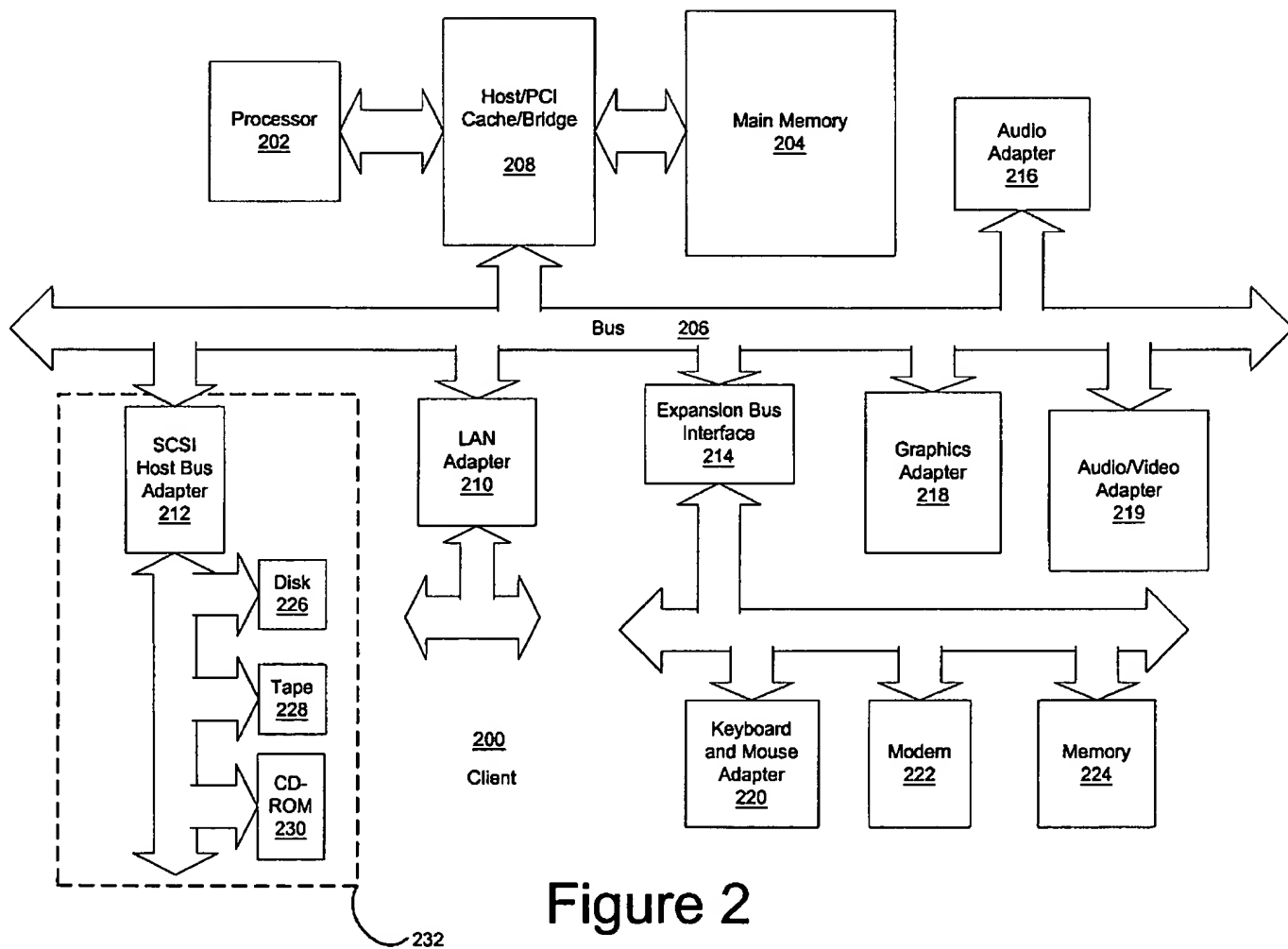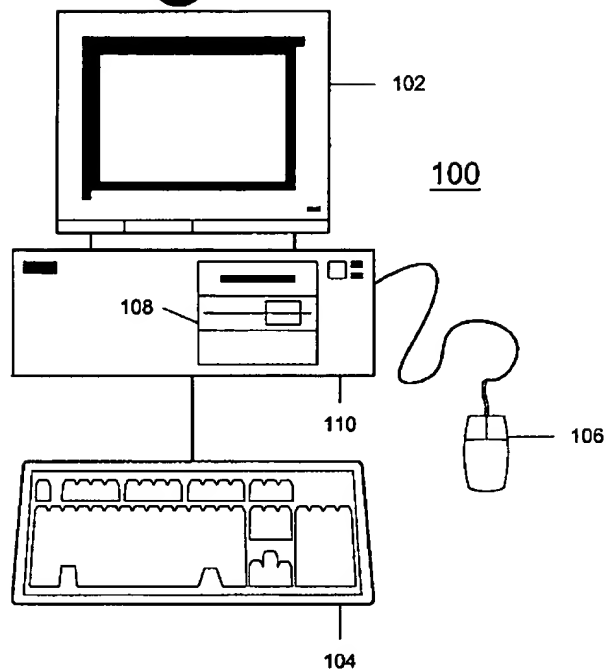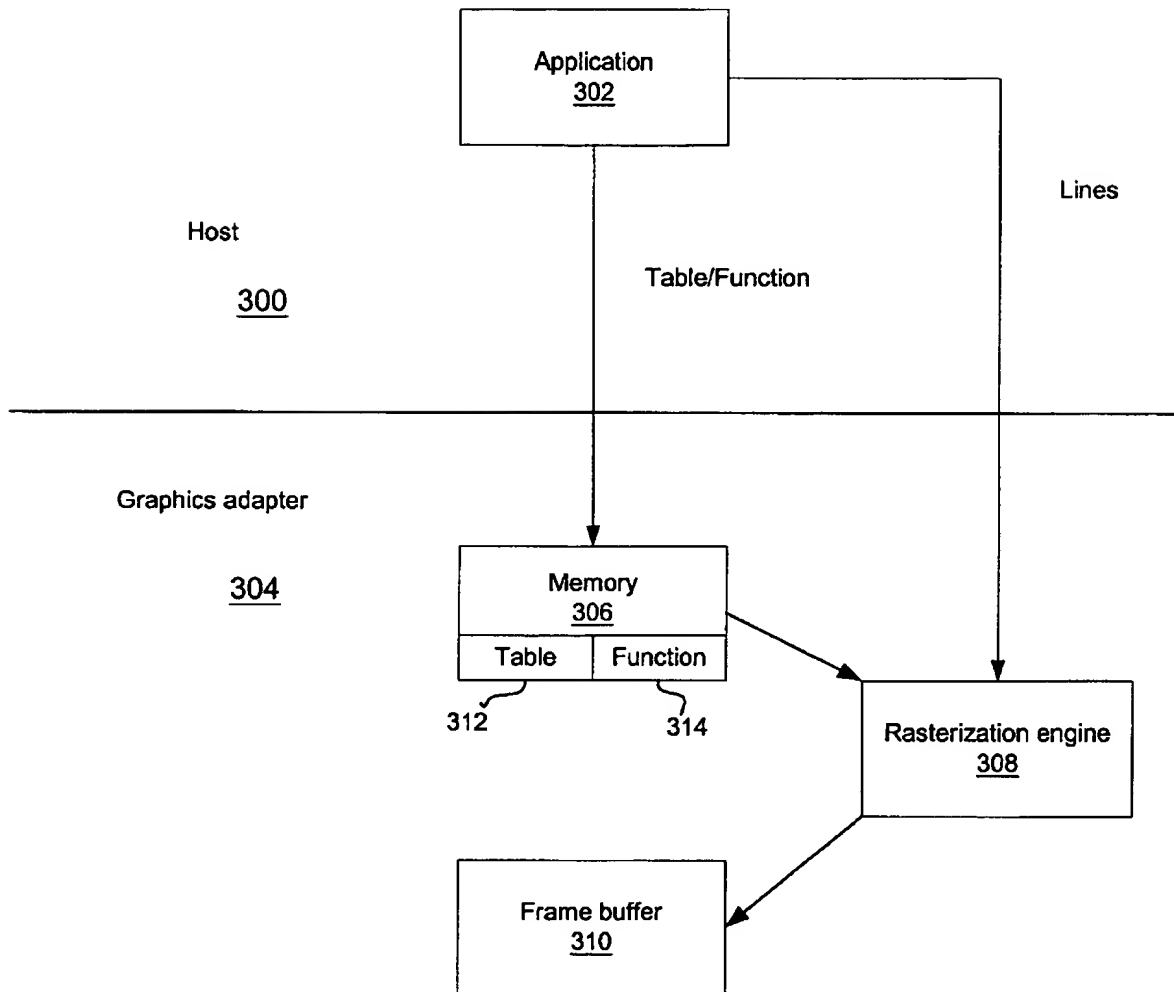
Brokenshire et al.
AUS920010010US1
Method and Apparatus for Generating
Gamma Corrected Antialiased Lines
Page 1 of 9

100

102

108

110

106

104

Processor **202**	Host/PCI Cache/Bridge **208**	Main Memory **204**	Audio Adapter **216**

Bus **206**

SCSI Host Bus Adapter **212**	LAN Adapter **210**	Expansion Bus Interface **214**	Graphics Adapter **218**	Audio/Video Adapter **219**

Disk **226**

Tape **228**

CD-ROM **230**

200

Client

Keyboard and Mouse Adapter **220**

Modem **222**

Memory **224**

232

Figure 2

Figure 3

Application
302

Lines

Host

Table/Function

300

Graphics adapter

304

Memory
306

Table	Function

312   314

Rasterization engine
308

Frame buffer
310
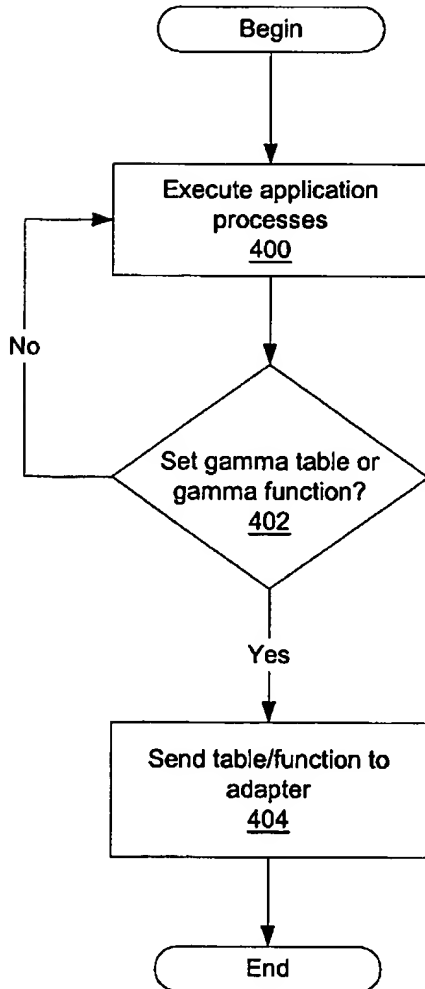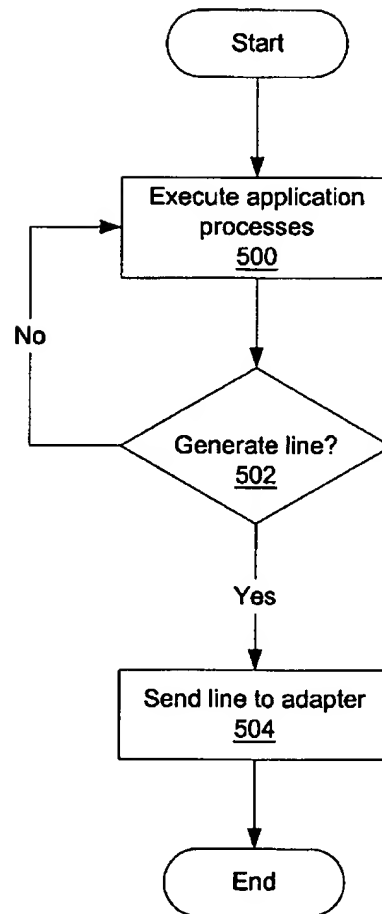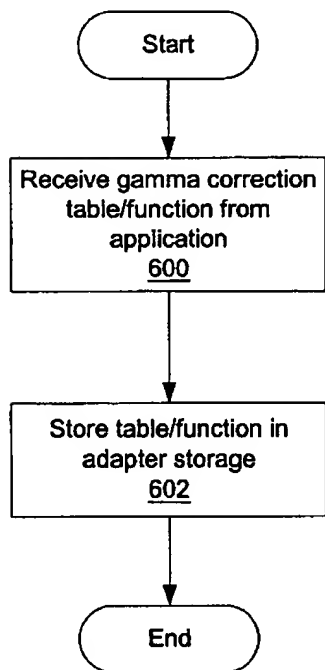
Figure 4

Brokenshire et al.
AUS920010010US1
Method and Apparatus for Generating
Gamma Corrected Antialiased Lines
Page 3 of 9

```
          ┌──────────────┐
          │    Begin     │
          └──────┬───────┘
                 │
                 ▼
          ┌──────────────┐
    ┌────►│   Execute    │
    │     │ application  │
    │     │  processes   │
    │     │     400      │
    │     └──────┬───────┘
    │            │
   No            ▼
    │         ╱──────╲
    │        ╱ Set    ╲
    │       ╱ gamma    ╲
    └──────◄ table or   ►
            ╲ gamma    ╱
             ╲function?╱
              ╲  402  ╱
               ╲────╱
                 │
                Yes
                 │
                 ▼
          ┌──────────────┐
          │ Send         │
          │ table/func-  │
          │ tion to      │
          │ adapter      │
          │     404      │
          └──────┬───────┘
                 │
                 ▼
          ┌──────────────┐
          │     End      │
          └──────────────┘
```

Figure 5

```
          ┌──────────────┐
          │    Start     │
          └──────┬───────┘
                 │
                 ▼
          ┌──────────────┐
    ┌────►│   Execute    │
    │     │ application  │
    │     │  processes   │
    │     │     500      │
    │     └──────┬───────┘
    │            │
   No            ▼
    │         ╱──────╲
    │        ╱        ╲
    └───────◄ Generate ►
             ╲ line?   ╱
              ╲  502  ╱
               ╲────╱
                 │
                Yes
                 │
                 ▼
          ┌──────────────┐
          │ Send line to │
          │   adapter    │
          │     504      │
          └──────┬───────┘
                 │
                 ▼
          ┌──────────────┐
          │     End      │
          └──────────────┘
```
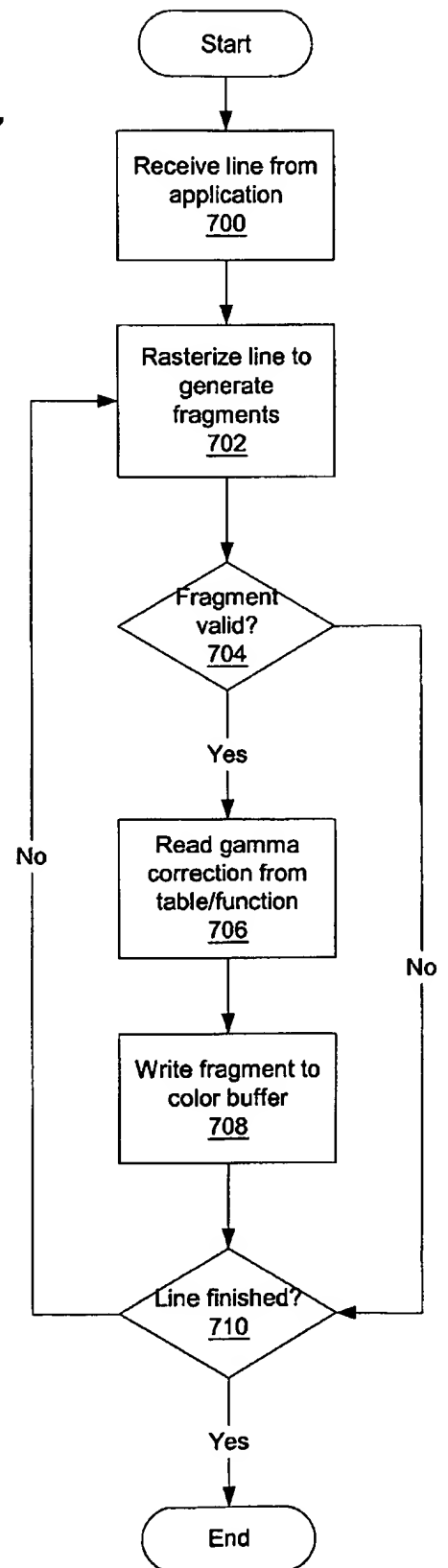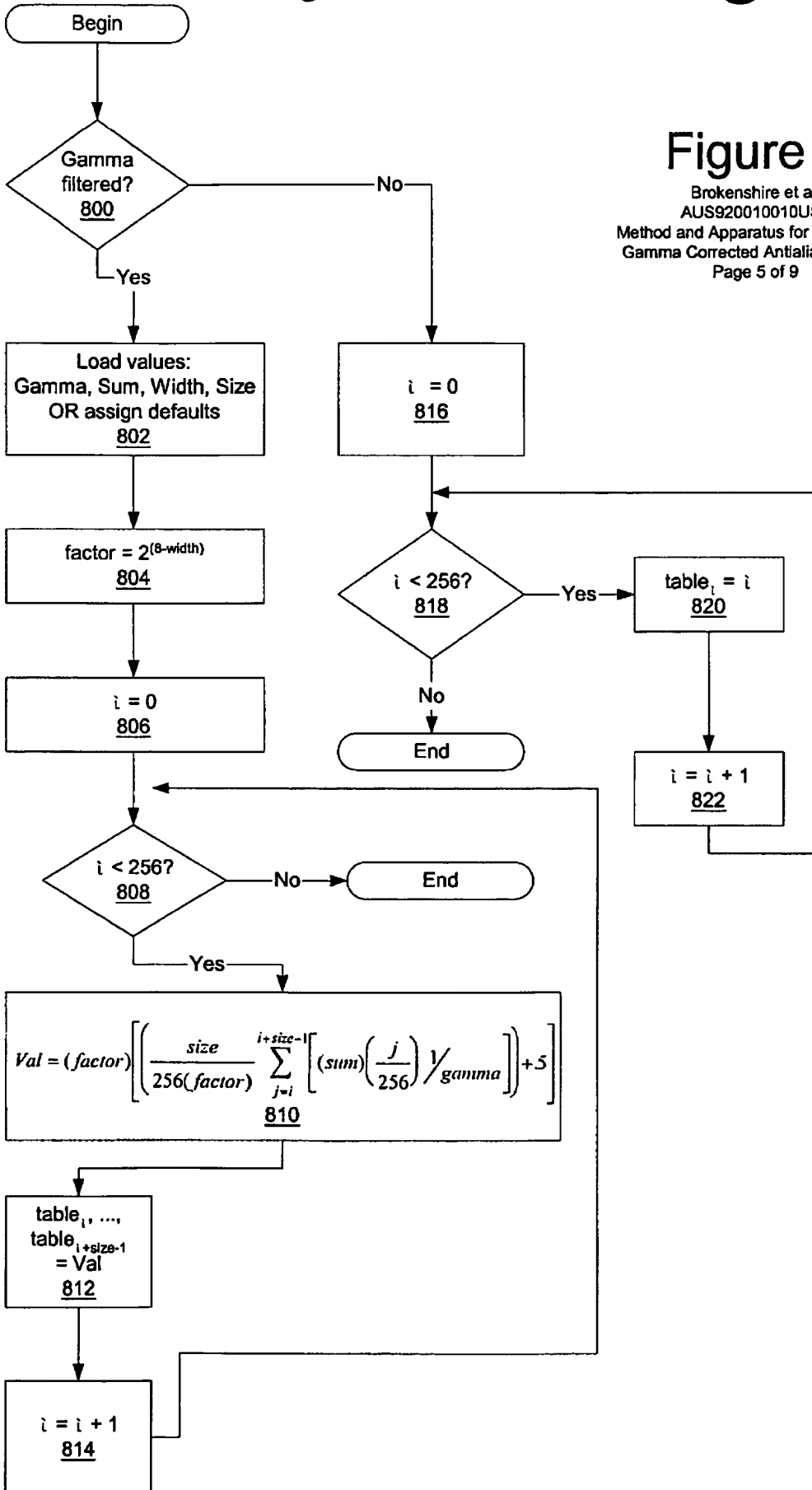
Figure 7

Brokenshire et al.
AUS920010010US1
Method and Apparatus for Generating
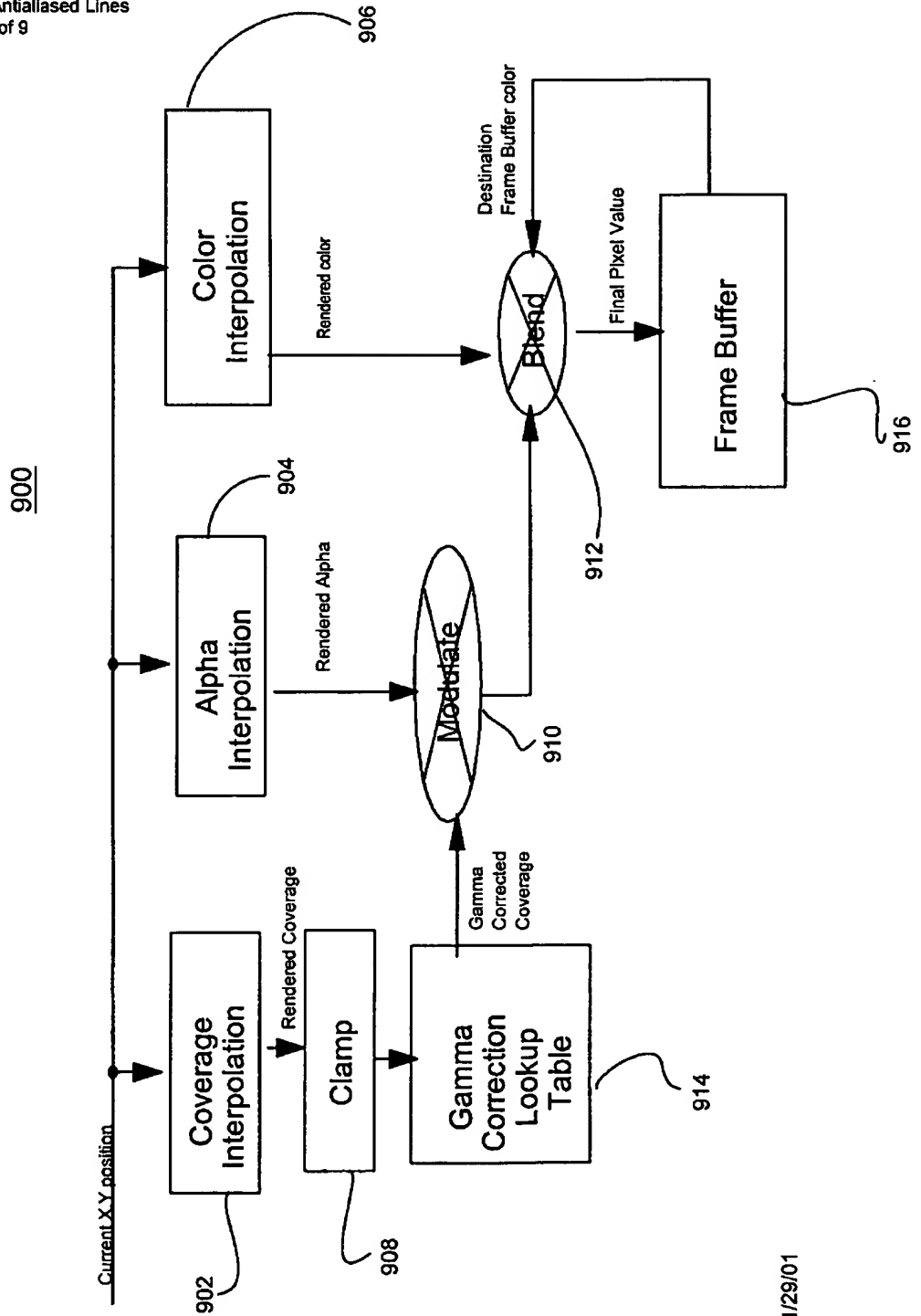Gamma Corrected Antialiased Lines
Page 4 of 9

Figure 6

Figure 6 flowchart:

Start
↓
Receive gamma correction table/function from application
600
↓
Store table/function in adapter storage
602
↓
End

Figure 7 flowchart:

Start
↓
Receive line from application
700
↓
Rasterize line to generate fragments
702
↓
Fragment valid?
704
— Yes ↓
Read gamma correction from table/function
706
↓
Write fragment to color buffer
708
↓
Line finished?
710
↓ Yes
End

704 — No → Line finished? 710
710 — No → Rasterize line to generate fragments 702

Figure 8

Begin

Gamma filtered? 800 — No

Yes

Load values: Gamma, Sum, Width, Size OR assign defaults 802

$\text{factor} = 2^{(8-width)}$ 804

$i = 0$ 806

$i < 256?$ 808 — No → End

Yes

$$Val = (factor)\left[\left(\frac{size}{256(factor)} \sum_{j=i}^{i+size-1}\left[(sum)\left(\frac{j}{256}\right)^{1/gamma}\right]\right)+5\right]$$
810

$table_i, ..., table_{i+size-1} = Val$ 812

$i = i + 1$ 814

$i = 0$ 816

$i < 256?$ 818 — Yes → $table_i = i$ 820

No

End

$i = i + 1$ 822

Figure 9

Brokenshire et al.
AUS920010010US1
Method and Apparatus for Generating
Gamma Corrected Antialiased Lines
Page 6 of 9

900

Current X.Y position

Color
Interpolation — 906

Rendered color

Alpha
Interpolation — 904

Rendered Alpha

Coverage
Interpolation — 902

Rendered Coverage

Clamp — 908

Gamma
Correction
Lookup
Table — 914

Gamma
Corrected
Coverage

Modulate — 910

Blend — 912

Destination
Frame Buffer color

Final Pixel Value

Frame Buffer — 916

JSL 1/29/01

```
if (env = getenv("_OGL_GAMMA_FILTER")){
/* Gamma filtered */
    float gamma;
    float sum;
    int    factor;
    int    width;
    int    size;

    gamma = 1.0;
    gamma = atof(env);

    width = 8;
    if (env = getenv("_OGL_GAMMA_TABLEWIDTH"))
       width = atoi(env);
    factor = (int)pow(2.0, (double) (8.0-width));

    sum = 256.0;
    if (env = getenv("_OGL_GAMMA_SUM"))
       sum = atof(env);

    size = 256;
    if (env = getenv("_OGL_GAMMA_TABLESIZE")){
       size = atoi(env);
       switch (size) {
       case 16:
        for (i=0; i<256; i+=16) {
          a = sum * pow((double)(i/256.0), (double)(1.0 / gamma));
          b = sum * pow((double)((i+1)/256.0), (double)(1.0 / gamma));
          c = sum * pow((double)((i+2)/256.0), (double)(1.0 / gamma));
          d = sum * pow((double)((i+3)/256.0), (double)(1.0 / gamma));
          e = sum * pow((double)((i+4)/256.0), (double)(1.0 / gamma));
          f = sum * pow((double)((i+5)/256.0), (double)(1.0 / gamma));
          g = sum * pow((double)((i+6)/256.0), (double)(1.0 / gamma));
          h = sum * pow((double)((i+7)/256.0), (double)(1.0 / gamma));
          i = sum * pow((double)((i+8)/256.0), (double)(1.0 / gamma));
          j = sum * pow((double)((i+9)/256.0), (double)(1.0 / gamma));
          k = sum * pow((double)((i+10)/256.0), (double)(1.0 / gamma));
          l = sum * pow((double)((i+11)/256.0), (double)(1.0 / gamma));
          m = sum * pow((double)((i+12)/256.0), (double)(1.0 / gamma));
          n = sum * pow((double)((i+13)/256.0), (double)(1.0 / gamma));
          o = sum * pow((double)((i+14)/256.0), (double)(1.0 / gamma));
          p = sum * pow((double)((i+15)/256.0), (double)(1.0 / gamma));
          AAFilterTable[i] = AAFilterTable[i+1] =
          AAFilterTable[i+2] = AAFilterTable[i+3] =
          AAFilterTable[i+4] = AAFilterTable[i+5] =
          AAFilterTable[i+6] = AAFilterTable[i+7] =
```

Figure 10A

1000

```
          AAFilterTable[i+8] = AAFilterTable[i+9] =
          AAFilterTable[i+10] = AAFilterTable[i+11] =
          AAFilterTable[i+12] = AAFilterTable[i+13] =
          AAFilterTable[i+14] = AAFilterTable[i+15] =
                 (Int)((((a + b + c + d + e + f +
                 (int)((((a + b + c + d + e + f +
                    g + h + ii + j + k + m +
                    n + o + p)/(16.0*factor)) + 0.5)*factor);

        }
        break;
        case 32;
          for (i=0; i<256; i+=8) {
            a = sum * pow((double)(i/256.0), (double)(1.0 / gamma));
            b = sum * pow((double)((i+1)/256.0), (double)(1.0 / gamma));
            c = sum * pow((double)((i+2)/256.0), (double)(1.0 / gamma));
            d = sum * pow((double)((i+3)/256.0), (double)(1.0 / gamma));
            e = sum * pow((double)((i+4)/256.0), (double)(1.0 / gamma));
            f = sum * pow((double)((i+5)/256.0), (double)(1.0 / gamma));
            g = sum * pow((double)((i+6)/256.0), (double)(1.0 / gamma));
            h = sum * pow((double)((i+7)/256.0), (double)(1.0 / gamma));
            AAFilterTable[i] = AAFilterTable[i+1] = AAFilterTable[i+2] =
            AAFilterTable[i+3] = AAFilterTable[i+4] = AAFilterTable[i+5] =
            AAFilterTable[i+6] = AAFilterTable[i+7] = (int)((((a + b + c + d + e + f + g + h)/(8.0*factor)) + 0.5)*factor);
        }
        break;
        case 64;
          for (i=0; i<256; i+=4) {
            a = sum * pow((double)(i/256.0), (double)(1.0 / gamma));
            b = sum * pow((double)((i+1)/256.0), (double)(1.0 / gamma));
            c = sum * pow((double)((i+2)/256.0), (double)(1.0 / gamma));
            d = sum * pow((double)((i+3)/256.0), (double)(1.0 / gamma));
            AAFilterTable[i] = AAFilterTable[i+1] =
            AAFilterTable[i+2] = AAFilterTable[i+3] =
               (int) ((((a + b + c + d)/4.0*factor)) + 0.5*factor);
        }
        break;
        case 128;
          for (i=0; i<256; i+=2) {
            a = sum * pow((double)(i/256.0), (double)(1.0 / gamma));
            b = sum * pow((double)((i+1)/256.0), (double)(1.0 / gamma));
            AAFilterTable[i] = AAFilterTable[i+1] =
               (int) ((((a + b)/2.0*factor)) + 0.5*factor);
        }
        break;
        case 256;
          for (i=0; i<256; i++) {
            AAFilterTable[i] =
               (int)((((sum * pow((double)(i/256.0), (double)(1.0 / gamma)))/factor) + 0.5)*factor);
        }
        break;
        }
}
```

Figure 10B

Brokenshire et al.
AUS920010010US1
Method and Apparatus for Generating
Gamma Corrected Antialiased Lines
Page 8 of 9

Figure 11

Brokenshire et al.
AUS920010010US1
Method and Apparatus for Generating
Gamma Corrected Antialiased Lines
Page 9 of 9

Assumptions: Floating point coverages are defined in the normalized 0.0
to 1.0 range in which 0.0 corresponds to no coverage and 1.0 corresponds
to full coverage. Fixed point coverages are defined in the range 0 to
size - 1.

```
float * GenFloatingPtGammaTable(int  size,
                                float gamma)
{
  int i;                                                         1100
  float *table;

  if (table = malloc(sizeof(float)*size)) {
    for (i=0; i<size; i++) {
      table[i] = (float)pow((double)i/(size-1), (double)(1.0/gamma));
    }
  }
  return (table);
}


int * GenFixedPtGammaTable(int  size,
                           float gamma)
{
  int i;
  int *table;
  float val;

  if (table = malloc(sizeof(int)*size)) {
    for (i=0; i<size; i++) {
      val = (float)pow((double)i/(size-1), (double)(1.0/gamma));
        table[i] = (int)((size-1) * val + 0.5);
    }
  }
  return (table);
}
```